Understanding Request and Response

The Foundation of Web Communication

What is HTTP Request and Response?

- **HTTP (HyperText Transfer Protocol)** is the foundation of web communication
- **Request**: Client (browser) asks for something from the server
- **Response**: Server sends back the requested data or status

```
Client (Browser) ----[Request]----> Server
Client (Browser) <---[Response]---- Server
```

Raw HTTP Request Format

When you visit a website, your browser sends something like this:

```
POST /api HTTP/1.1
Host: localhost:8000
Content-Type: application/json

{
  "student_id": 1
}
```

Notice the space between headers and body.

Components:

- **Request Line**: Method + Path + HTTP Version
- **Headers**: Metadata about the request
- **Body**: Data (for POST/PUT requests)

Raw HTTP Response Format

The server responds with something like this:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 285
Date: Sat, 02 Aug 2025 10:30:00 GMT
Server: Apache/2.4.41

{
    "success": true,
    "message": "Welcome to Simple PHP API",
    "data": {
        "name": "Simple PHP API for Education",
        "version": "1.0"
    }
}
```

Components:

- **Status Line**: HTTP Version + Status Code + Status Message
- **Headers**: Metadata about the response
- **Body**: The actual content

Why Manual Parsing is Complex

- Web servers must interpret and parse incoming client requests to generate appropriate responses.
- This task is challenging because manually parsing these requests involves the following, all of which increase the likelihood of mistakes and inefficiency.

1. **Format Complexity**

```
GET /search?q=hello+world&lang=en HTTP/1.1\r\n
Host: example.com\r\n
User-Agent: Mozilla/5.0...\r\n
\r\n
```

- Must handle `\r\n` line endings
- Parse query parameters manually
- Handle URL encoding (`+` = space, `%20` = space)

2. **Header Parsing Challenges**

```python
# Manual parsing example (Python-like pseudocode)
def parse_headers(raw_request):
    lines = raw_request.split('\r\n')
    request_line = lines[0].split(' ')
    method = request_line[0]
    path = request_line[1]

    headers = {}
    for line in lines[1:]:
        if line == '':  # Empty line separates headers from body
            break
        key, value = line.split(': ', 1)
        headers[key.lower()] = value

    return method, path, headers
```

Problems:

- Error-prone string manipulation
- Must handle edge cases (malformed requests)
- Security vulnerabilities if not careful

3. **URL and Query Parameter Parsing**

```
/search?name=John%20Doe&age=25&city=New%20York
```

Manual parsing required:

- Split path from query string
- Decode URL encoding (`%20` → space)
- Parse key-value pairs
- Handle arrays: `?colors=red&colors=blue`
- Handle special characters

- **Path**:
  `/search`
  -> The endpoint/resource requested.
- **Query String**:
  `name=John%20Doe&age=25&city=New%20York`
  -> Key-value pairs after `?`.
- Each item is separated by `&`
  - `name=John%20Doe` → `name = "John Doe"`
  - `age=25` → `age = 25`
  - `city=New%20York` → `city = "New York"`

1. **Content Type Handling**

- **JSON**: `{"name": "John"}`
- **Form data**: `name=John&age=25`
- **Multipart**: File uploads with boundaries
- **XML**: `<user><name>John</name></user>`

Result

- Hundreds of lines of complex code
- Security vulnerabilities
- Maintenance nightmare
- Reinventing the wheel

We Need Web Frameworks

Problem: Manual parsing is complex and error-prone

Solution: Programming languages and frameworks provide:

- ✅ **Built-in parsers** for HTTP requests/responses
- ✅ **Security handling** (input validation, sanitization)
- ✅ **Abstraction layers** (simple variable access)
- ✅ **Standard patterns** (routing, middleware)

There are many frameworks that can help us to manage this complexity.

PHP for Managing HTTP Complexity

In this course, we use **PHP** to manage HTTP complexity:

- Automatic parsing of **query strings** (`$_GET`)
- Automatic parsing of **form data / JSON** (`$_POST` , `php://input`)
- Superglobals (`$_SERVER` , `$_REQUEST`) for headers & environment
- Built-in functions to simplify request/response handling

PHP Framework convert complex raw HTTP

```
GET /api?name=John&age=25 HTTP/1.1
Host: localhost:8000
```

Into simple variable access through parsing:

```
$method = $_SERVER['REQUEST_METHOD'];  // "GET"
$name = $_GET['name'];                 // "John"
$age = $_GET['age'];                   // "25"
```